

## Client Server Registration Protocol

The Client-Server protocol involves these following steps:

1. Login
2. Discovery phase

- 🚩 User (Alice or Bob) has  $K_S$
- 🚩 Server (S) has **hash[PW<sub>A</sub>]**. The passwords hashes are stored in order to prevent the server from being vulnerable to database reading.
- 🚩  $K_S$ : Public key of server     $K_A$ : Public key of Alice
- 🚩  $PW_A$ : Password of Alice     $K_{AS}$ : Secret key shared between Alice and the Server
- 🚩  $IP_A$ : IP address of Alice     $P_A$ : Port of Alice

- 🚩 Cookie(C): (Sent by Server) 32-bit Random Number **XOR** Current time.
- 🚩 Cookie(C'): (Sent by Client) C **OR** (hash[PW<sub>A</sub>] **OR**  $K_S$ )

### Login

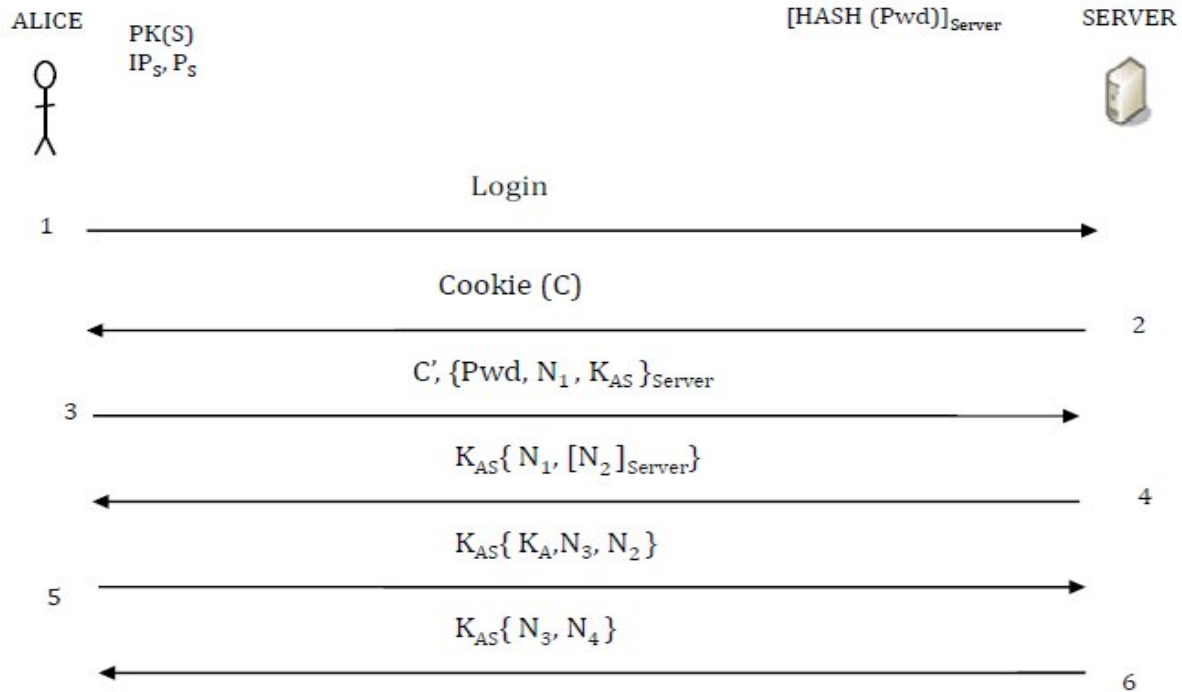


Figure 1.1

In this phase the User authenticates itself to the server and vice-versa. So we provide mutual authentication by doing this.

1. In this step of Authentication, the User sends a Login message to the Server.
2. Then the Server sends back a Cookie(C) to the Client.
3. Client then performs some computations on the received cookie to form C'. It encrypts the Password,  $N_1$  and the shared key ( $K_{AS}$ ) with the public key of the server. Client sends this along with the Cookie (C').
4. The Server sends back the Nonce ( $N_1$ ) sent by the Client and a new Nonce ( $N_2$ ), signed by the Server. This is done by the server to prove that it knows the shared secret and the Nonce( $N_1$ ).
5. Then once the Client receives this it sends its Public key ( $K_A$ ) to the server along with a nonce.
6. Then the server replies back and this message just act as an acknowledgement to the receipt of Public key of the Client.

### Discovery Phase

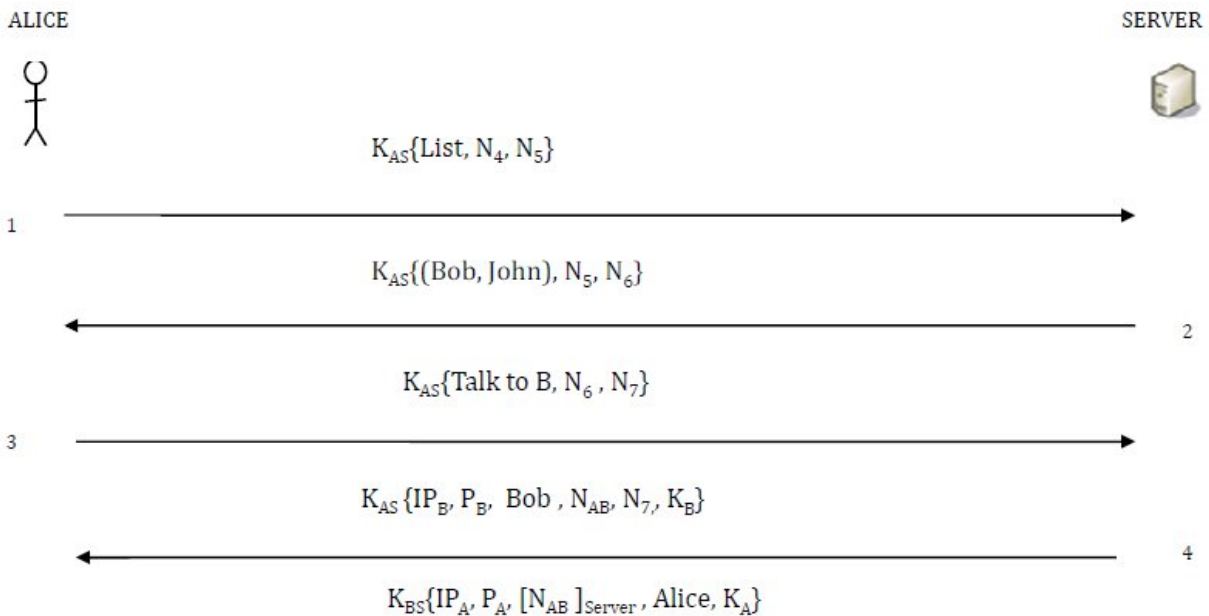


Figure 1.2

In this Phase, Client discovers the list of online users from the server. And then the Client selects a particular User to talk to, and obtains the credentials needed to talk to that user from the Server.

1. In this step, the Client clicks on the "List" command and the client sends Nonce ( $N_4$ ) he receives in the Login Phase of the protocol encrypted with the Secret key shared between the Client and the server.
2. Then the Server sends the List of online users encrypted with the Shared key ( $K_{AS}$ ).
3. Client picks a user he wants to talk to and then sends it to the server encrypted with the Shared key.
4. Server matches its database and sends the credentials to the respective clients. The credentials include the IP address ( $IP_B$ ), Port ( $P_B$ ), and the Public Key of the User ( $K_B$ ).

## ISSUES

### Mutual Authentication

Mutual authentication is provided between the Client (Alice or Bob) and the Server as follows:

In the Login phase, this is achieved when the Server sends a Cookie(C) to the Client. The Client returns back a modified version of the cookie (i.e., C'). Once the Server receives the C' from the client, it tries to compute C' in the same manner in which the client computes it and verifies it with the received quantity. By this exchange Server authenticates the Client. This is the first level of Server authenticating the client. It also authenticates the client based on the password that he enters.

Then in the fourth message of the Login Phase, the server signs the Nonce ( $N_2$ ) with its own private key. By this exchange the Client authenticates the Server as it is the only entity to know the private key.

Hence these different levels provide mutual authentication between the Client and the server.

### Replay Attacks

This protocol is not vulnerable to Replay attacks as we maintain the freshness of the messages between the communicating entities by generating random values called Nonce in each message and we expect that Nonce back in some message or the other. So, any intruder listening to the conversation between the communicating parties cannot replay the messages as they can easily detect and differentiate between a replayed and fresh message.

### Resilience to Denial of Service

This protocol is resilient to denial of service attack as we delay the connection establishment and registration until the message 4. So, there are many levels of

authentication we provide in order to be resilient to DOS attack. (i.e., the first level of Authentication is by using the cookies and next by using the Password)

### **Syn-flooding**

We prevent the DOS attack in this case by using cookies, which the Server sends in message 2 in the Login phase, and expects a modified cookie (C') which it later checks to authenticate the Client. So, any intruder claiming to be a authorized client cannot do anything as he cannot provide a valid C'.

### **Smurf attack or Fraggle attack**

Since, we make use of TCP connections for the connection establishment between the Client and the Server we do not broadcast any messages. So, it is resilient to Smurf attacks.

If we make use of UDP connections then we can make the system to resilient to Fraggle attacks if we can use a filtering policy at the firewall to drop all the bogus packets. We can disable the service port (port # 7) and chargen port( port # 19), so that fraggle attacks can be avoided. This is done as these ports are used to exploit this kind of attacks

### **Ping Flooding**

To reduce the effects of a ping flood, the server can use a firewall to filter the incoming ICMP Echo Request packets entirely, or if a large number of requests are received at one time. Refusing to send ICMP Echo Reply packets produces two benefits:

1. Less bandwidth is wasted by not answering these packets.
2. It is more difficult for the attacker to measure the effectiveness of the attack.

## Client - Client Communication Protocol

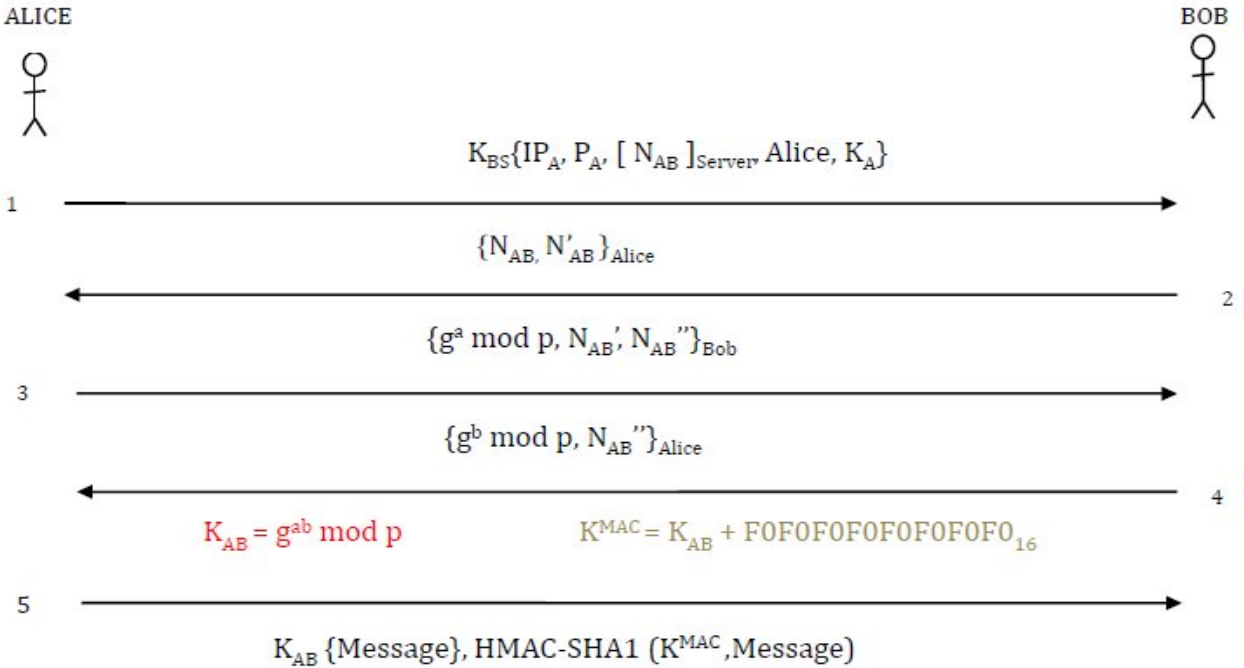


Figure 1.3

Client-Client communication protocol has the following layout:

After Client-Server registration protocol has completed, the server will have the following information in its memory:

- IP-address
- Port
- Client's Name
- Public Key of registered Client's.

In order to understand how a secure channel is established between clients, let's consider Alice and Bob as two clients registered with the server sharing the following information:

- $IP_A$  : IP address of Alice &     $IP_B$  : IP address of Bob
- $P_A$  : Port Number of Alice &     $P_B$  : Port number of Bob
- $N_{A''}$  : Nonce for Alice &         $N_{B''}$  : Nonce for Bob
- $A$  : Alice's Identity &          $B$  : Bob's Identity
- $K_A$  : Public Key of Alice &      $K_B$  : Public Key of Bob

## Session Key Establishment

1. In this step the Alice just forwards the message it receives from the Server to Bob. Bob is the only entity which can decrypt this message as it is encrypted by the Secret key shared between it and the server.
2. Bob then sends a new Nonce ( $N_{AB}$ ) and  $N_{AB}$  encrypted by Alice's public key
3. Once each of the Communicating parties know with whom they are talking to. Alice decrypts the nonce with its session key and verifies  $N_{AB}$  received with previous message sent by Alice, if everything is proper Alice chooses a random number "a" and initiates diffie-hellman by generating  $g^a \bmod p$  encrypts it with public key of Bob  $K_B$  and sends it to Bob.
4. Bob receives  $g^a \bmod p$ , chooses random "b" generates  $g^b \bmod p$  encrypts it with public key of Alice and replies it to Alice.
5. Now the session key between Alice and Bob is given by  $K_{AB} = g^{ab} \bmod p$ . They compute the MAC key as shown in the figure.

Going forward data generated by Alice or Bob is appended with Nonce which will be relayed back to the originator in the subsequent interaction this is done in order to prevent attackers from replaying messages and the randomness of nonce makes it difficult for the attacker to guess as well.

## Mutual Authentication

Authentication is carried out to ensure that both the clients can be trusted before communication can begin and they are legitimately registered with the server.

Initially the message sent by the Server to Alice is just relayed to Bob, and Bob comes to know that this message is coming from Alice, as in the discovery message Alice is the only entity who is able to get this message from the server, and it also verifies the  $IP_A$  from which the message is originating. Alice authenticates Bob by checking at the message 2 as the Nonce ( $N_{AB}$ ) comes back to Alice, and it knows Bob is the only entity who could have sent it.

In this way Mutual authentication is achieved between two clients before the communication can begin.

## Integrity

Integrity is achieved through the use of MAC code which will be sent along with the data, which is a digest of the data and can be verified to make sure that it's not been tampered in any way.

Why a different key for MAC Code?

This is to make sure that if the session key is compromised the attacker won't be able to provide MAC code with a proper key hence this will give a chance identify that the session has been high jacked by an intruder.

## Confidentiality

The method to keep data confidential is to encrypt the data this is done by carefully choosing the encryption algorithms.

1. RSA Asymmetric Algorithm with (1024 bit key size) this was chosen to due to the following reasons:
  - ✘ Brute force attack requires an exponential amount of overhead, the security of RSA is based on the difficulty of factoring.
  - ✘ Sender can only encrypt the message, private key is only known to the receiver.
  - ✘ Factoring 1024 bit key is really difficult, since hack is based on factoring.
  - ✘ RSA encryption is very slow hence only the secret key is encrypted using RSA.
2. AES Symmetric algorithm to encrypt the file with a secret key of size 256 bits.
  - ✘ Computationally less intensive than public-key cryptography.
  - ✘ Easy to implement in hardware as well as software.
  - ✘ Can be implemented as block ciphers or stream ciphers.
  - ✘ Allows for variable block size and variable key sizes 128,192,256.
  - ✘ This algorithm is found to be more secure when compared to DES, IDEA or 3DES the other symmetric key algorithms.
  - ✘ It's difficult to decrypt key of 256 bits by brute force hence larger key size.
3. CBC Mode
  - ✘ Allows for encryption of data in blocks of size 64-bits
  - ✘ If same blocks repeat in the plain text, it will not be repeated in cipher text.
  - ✘ CBC generates its own random numbers from a large set of numbers, it takes the previous block of cipher text and uses that as the random number that will be XOR'd with the next plaintext, hence anyone tampers with cipher text, the integrity of the cipher text can be found out easily.
4. Digital Signatures
  - ✘ Ensures that the party sending knows the private key and is a legitimate user hence used in authentication.
5. MAC Code
  - ✘ The MAC digest is generated using HMAC-SHA1 which is known to be very secure against brute force attack and hence providing integrity.

## Reflection Attack

As we can see that the initiator authenticates first to the receiver see message 1 and 2, hence defending against possible reflection attacks from the intruder. The presence of Nonce adds to the difficulty hence preserving the freshness of the messages.

## Replay Attack

Intruder who is recording the conversation between Alice and Bob might try to replay the previously recorded messages in between authentication process or while establishing a session key. The system designed protects against these kind of replay attacks by including a nonce in each of the messages (see  $N_{AB}$ ,  $N_{AB'}$ ,  $N_{AB''}$  between client's) which will be replied back to ensure the freshness of messages being exchanged. If any malicious message from previous recordings were replayed the verification of the nonce identifies its freshness and can be discarded if it doesn't match. Nonce is encrypted in the messages hence intruder can't predict the value of nonce easily since it will be generated from a fairly large random space.

## Resilience to Denial of Service

Denial of service attack is the most difficult attack to be protected against the design of the current protocol ensures solutions to most of the DoS attacks thought of by the intruder.

1. Message Flooding: From message 1, the server gives the initiator (Alice), credentials of Alice to Bob encrypted by the secret key shared between Bob and the Server. This is done to ensure that other party it is receiving the data from is originating from the same source, if the originating entity is a spoofed address of Alice which sends a flood of messages to Bob, if the messages doesn't match Bob keeps a count of the number of trails attempted by the originating IP address if it exceeds with failures then Bob drops the packets originating from the source IP address then onwards not accepting them for a certain period of time hence the spoofer will be kept waiting for any replies thereby protecting against DoS.
2. Smurf attacks (TCP) and Fraggle attacks (UDP): This is flooding of the channel with broadcast packets; the protocol designed uses the underlying TCP to establish connections hence any broadcast packets can be made to drop at the firewall itself since it doesn't provide any purpose in our protocol.

In case the protocol designed uses UDP to establish connections any broadcast packets can be filtered at the Firewall, and protect the system from Fraggle attacks.

## Client - Client Connection Release Protocol

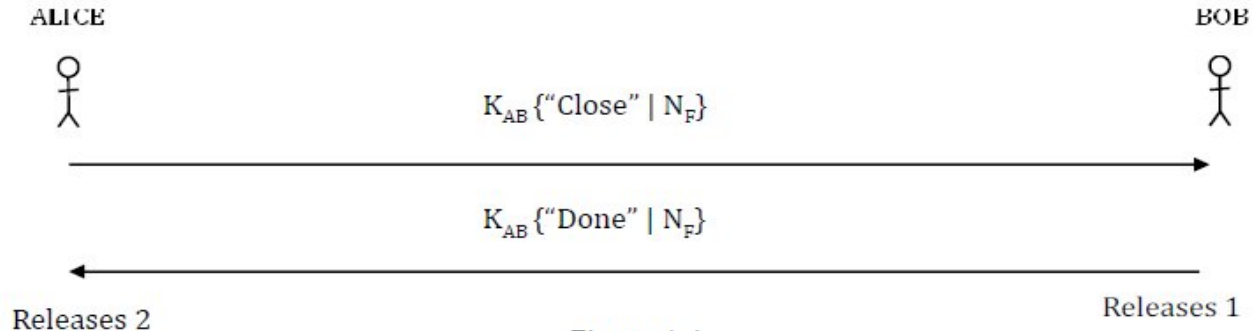


Figure 1.4

OR

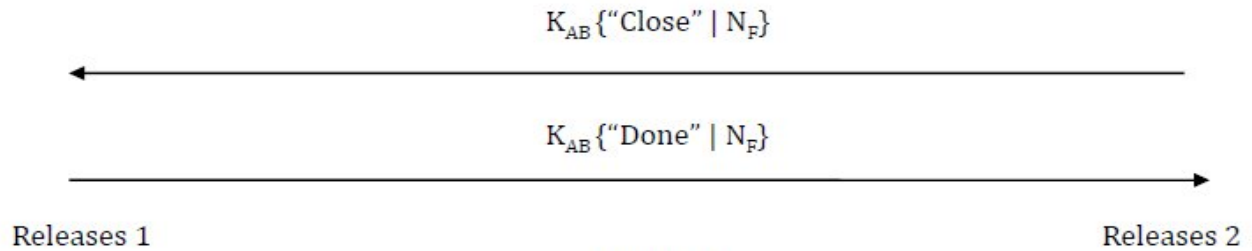
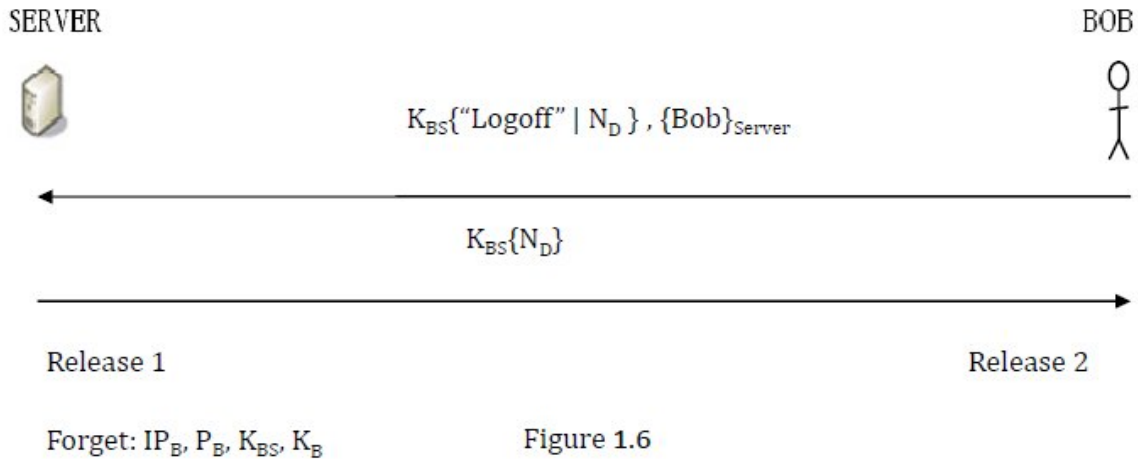


Figure 1.5

The termination of the communication could be initiated by any of the two clients in the conversation.

- ❖ Alice (say) initiates the termination. Alice sends the simple "Reset" message and a Nonce ( $N_F$ ) encrypted by the shared key between Alice and Bob. Bob in return will the "Done" message in the same format as the "Reset" message was received.
- ❖ This process could be initiated by any of the users and same steps will be followed as explained above. Thus in the end, both clients releases the resource reserved by them for each other.

## Logoff Protocol



This protocol will be followed when the client (here say Bob) wants to terminate the chat. The above messages will be sent by the client code running at Bob to the server, so that the resources used by the server, for Bob, will be forgotten.

- ❖ Bob sends a "Logoff" message to the Server, concatenated with Nonce ( $N_D$ ) and its own identity "Bob". This whole message is encrypted by the secret key shared between the Client and the Server.
- ❖ Server sends the Nonce of Bob ( $N_D$ ) encrypted by the shared key of Bob and Server. This proves to Bob that it is coming from the intended Server.

Once Server sends the second message it forgets or all the resources allocated to Bob. If in case Bob doesn't receive the second message, then Bob resends message 1 but Server doesn't have any information related to Bob anymore. So, the server just sends a message saying that it is not having information related to Bob. Bob then confirms that it has been logged out of the Server and then it just releases its resources pertaining to the server.

## Identity Hiding

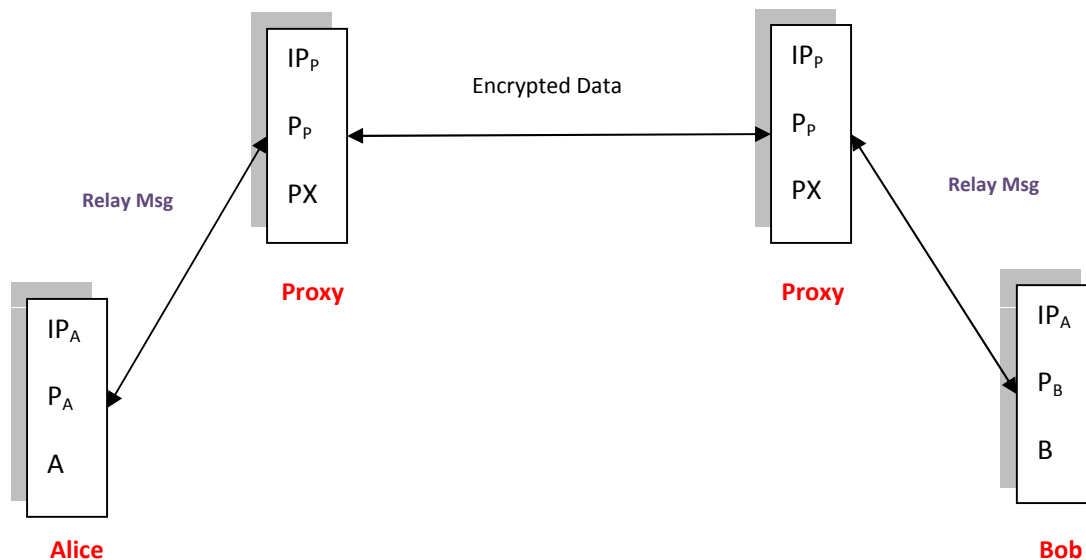


Figure 1.5

Identity hiding is to conceal the information of originating entity or end point hiding, even though the identity information is secured by using powerful encrypting algorithms and diffe-hellman algorithm, we need to conceal the IP addresses as well which makes up the identity of Alice and Bob. Any intruder can find out originating source of Alice and Bob by using wire shark and can perform DoS attacks in order to prevent this of attack we propose the following techniques which helps prevent these.

1. IPSEC: This provides an encryption of IP datagram concealing the IP address of the originating as well as the destination source this can be done by having an underlying architecture built to help facilitate IPSEC implementation.
2. Proxy: As we find having an IPSEC infrastructure is expensive in our domain of implementation we propose another technique which can be easily realized by Figure 1.5 where in we have proxy configured with a different IP address at each of the client sites, any messages that are exchanged between the clients here Alice and Bob is relayed through the proxy, this prevents finding the original IP address of Alice and Bob since proxy can forward the messages if received from the other party internally to Alice or Bob. Intruder will be tricked in to believing that originating sources are proxies hence we can prevent DoS attack intended directly to Alice or Bob.
3. The techniques that we propose to use for identity hiding is either using the PREROUTING chain in the NAT table, so that the packets leaving the firewall will have their source address changed and thus not revealing the true IP address and thus hiding the identity.

## Questions

**Does your system protect against the use of weak passwords? Discuss both online and offline dictionary attacks.**

Yes our system is resilient to online attacks as the information sent by the Client can only be decrypted by the Server and no one else.

Our protocol is also resilient to offline dictionary attack as we store only the hashed value of the password at the server, which is very difficult to break.

**Is your design resistant to denial of service attacks?**

Yes, check the resilience to Denial of service in the Client-Server registration and Client-Client Protocol.

**To what level does your system provide end-points hiding, or perfect forward secrecy?**

Yes, the system provides perfect forward secrecy as we use Diffe-Hellman for the Session key exchange between the Client-Server and the Client-Client.

**If the users do not trust the server can you devise a scheme that prevents the server from decrypting the communication between the users without requiring the users to remember more than a password? Discuss the cases when the user trusts (vs. does not trust) the application running on his workstation.**

## Trust

When the User trusts his/her workstation then he just stores the key Shared keys between the User and Server ( $K_{AS}$  or  $K_{BS}$ ) and shared key between the two clients  $K_{AB}$  at the workstation, so that it can encrypt and decrypt whatever information is sent and reduce the burden on the user.

## Doesn't Trust

When the User doesn't trust his/her workstation then even if the workstation is compromised then the intruder will not be able to decrypt or encrypt anything as the workstation doesn't have the Shared Secrets.

**If the users do not trust the server can you devise a scheme that prevents the server from decrypting the communication between the users without requiring the users to remember more than a password?**

The information available with the server is the IP address, Port and Public key of the registered entities. If the underlying server cannot be trusted either Alice or Bob who registers with the un-trusted server gives only its IP address, Port, Public key and a nonce which is of importance to registered entities during client-client authentication.

Let's assume if the server can impersonate as Bob to Alice, server can only encrypt with Bob's public key since it doesn't have any information of its private key while establishing a session key using diffe-hellman, the message sent  $\{g^a \bmod p\} K_B$  can't be decrypted with just the public key of Bob hence further process gets stalled and cannot complete the session key establishment.

Let's assume if the server can impersonate as Alice to Bob, here also server won't be able to establish a session key with Bob only knowing Alice's public key as before.

As we can see in message 3,4, and 5 MAC key as well as session key is generated by communicating entities hence server can't decrypt the communication because keys are confined and known to only the clients.

Hence we can with absolute confidence say that the information available at server's disposal doesn't serve any purpose in decrypting communication between clients.